JBoss Enterprise Application Platform Continuous Delivery 15

How to Set Up SSO with Kerberos

For Use with JBoss Enterprise Application Platform Continuous Delivery 15

Last Updated: 2019-01-25

JBoss Enterprise Application Platform Continuous Delivery 15 How to Set Up SSO with Kerberos

For Use with JBoss Enterprise Application Platform Continuous Delivery 15

Legal Notice

Abstract

The intent of this guide is to explore the topic of single sign-on (SSO) with Kerberos within Red Hat JBoss Enterprise Application Platform as well as provide a practical guide for setting up SSO with Kerberos in JBoss EAP. Essentially this guide is providing a deeper dive into what SSO with Kerberos is as well as how to set up and configure it within JBoss EAP. Before reading this guide, users should read through the Security Architecture document for Red Hat JBoss Enterprise Application Platform and have a solid understanding of the SSO and Kerberos information presented in that document. This guide also makes use of the JBoss EAP CLI interface for performing configuration changes. For more information on using the CLI for both standalone JBoss EAP instances as well as JBoss EAP domains, see the JBoss EAP Management CLI Guide. When completing this guide, readers should have a solid, working understanding of SSO and Kerberos, how it relates to JBoss EAP, and how to configure it.

Table of Contents

PREFACE

This document is intended for use with the JBoss Enterprise Application Platform continuous delivery release 15, which is a Technology Preview release available in the cloud only.

Some features described in this document might not work or might not be available on Red Hat OpenShift Online and Red Hat OpenShift Container Platform. For specific details about the feature differences in the JBoss EAP CD release, see the Release Limitations section in the *JBoss EAP Continuous Delivery 15 Release Notes*.

IMPORTANT

This continuous delivery release for JBoss EAP is provided as Technology Preview only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

CHAPTER 1. SSO WITH KERBEROS DEEPER DIVE

1.1. WHAT ARE SSO AND KERBEROS?

A basic background of single sign-on (SSO) and Kerberos is provided in the Single Sign-On section of the JBoss EAP *Security Architecture* guide.

1.2. KERBEROS COMPONENTS

Kerberos itself is a network protocol that enables authentication for users of client/server applications through the use of secret-key cryptography. Kerberos is usually used for authenticating desktop users on networks, but through the use of some additional tools, it can be used to authenticate users to web applications and to provide SSO for a set of web applications. This essentially allows users who have already authenticated on their desktop network to seamlessly access secured resources in web applications without having to reauthenticate. This concept is known as desktop-based SSO since the user is being authenticated using a desktop-based authentication mechanism, and their authentication token or ticket is being used by the web application as well. This differs from other SSO mechanisms such as browser-based SSO, which authenticates users and issues tokens all through the browser.

The Kerberos protocol defines several components that it uses in authentication and authorization:

Tickets

A *ticket* is a form of a security token that Kerberos uses for issuing and making authentication and authorization decisions about principals.

Authentication Service

The *authentication service* (AS) challenges principals to log in when they first log into the network. The authentication service is responsible for issuing a ticket granting ticket (TGT), which is needed for authenticating against the ticket granting service and subsequent access to secured services and resources.

Ticket Granting Service

The *ticket granting service* (TGS) is responsible for issuing service tickets and specific session information to principals and the target server they are attempting to access. This is based on the TGT and destination information provided by the principal. This service ticket and session information is then used to establish a connection to the destination and access the desired secured service or resource.

Key Distribution Center

The *key distribution center* (KDC) is the component that houses both the TGS and AS. The KDC, along with the client, or principal, and server, or secured service, are the three pieces required to perform Kerberos authentication.

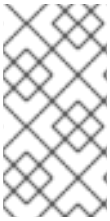
Ticket Granting Ticket

A *ticket granting ticket* (TGT) is a type of ticket issued to a principal by the AS. The TGT is granted once a principal successfully authenticates against the AS using their username and password. The TGT is cached locally by the client, but is encrypted such that only the KDC can read it and is unreadable by the client. This allows the AS to securely store authorization data and other information in the TGT for use by the TGS and enables the TGS to make authorization decisions using this data.

Service Ticket

A *service ticket* (ST) is a type of ticket issued to a principal by the TGS based on their TGT and the

intended destination. The principal provides the TGS with their TGT and the intended destination, and the TGS verifies the principal has access to the destination based on the authorization data in the TGT. If successful, the TGS issues an ST to the client for both the client as well as the destination server which is the server containing the secured service or resource. This grants the client access to the destination server. The ST, which is cached by the client and readable by both the client and server, also contains session information that allows the client and server to communicate securely.



NOTE

There is a tight relationship between Kerberos and the DNS settings of the network. For instance, certain assumptions are made when clients access the KDC based on the name of the host it is running on. As a result, it is important that all DNS settings in addition to the Kerberos settings are properly configured to ensure that clients can connect.

1.3. ADDITIONAL COMPONENTS

In addition to the Kerberos components, several other items are needed to enable Kerberos SSO with JBoss EAP.

1.3.1. SPNEGO

Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO) provides a mechanism for extending a Kerberos-based single sign-on environment for use in web applications.

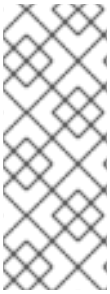
SPNEGO is an authentication method used by a client application to authenticate itself to the server. This technology is used when the client application and server are trying to communicate with each other, but neither are sure of the authentication protocol the other supports. SPNEGO determines the common GSSAPI mechanisms between the client application and the server and then dispatches all further security operations to it.

When an application on a client computer, such as a web browser, attempts to access a protected page on the web server, the server responds that authorization is required. The application then requests a service ticket from the Kerberos KDC. After the ticket is obtained, the application wraps it in a request formatted for SPNEGO, and sends it back to the web application, through the browser. The web container running the deployed web application unpacks the request and attempts to authenticate the ticket. Upon successful authentication, access is granted.

SPNEGO works with all types of Kerberos providers, including the Kerberos service included in Red Hat Enterprise Linux and the Kerberos server, which is an integral part of Microsoft Active Directory.

1.3.2. JBoss Negotiation

JBoss Negotiation is a framework that ships with JBoss EAP that provides an authenticator and JAAS login module to support SPNEGO in JBoss EAP. JBoss Negotiation is only used with the legacy **`security`** subsystem and legacy core management authentication. For more information on JAAS login modules, please see the Declarative Security and JAAS and Security Domains sections of the JBoss EAP *Security Architecture* guide.



NOTE

When using JBoss Negotiation to secure certain applications, such as REST web services, one or more sessions may be created and left open for the timeout period, which defaults to 30 minutes, when a client makes a request. This differs from the expected behavior of securing an application using basic authentication, which would leave no open sessions. JBoss Negotiation is implemented to use sessions to maintain the state of the negotiation/connection so the creation of these sessions is expected behavior.

1.4. KERBEROS INTEGRATION

Kerberos is integrated with many operating systems including Linux distributions such as Red Hat Enterprise Linux. Kerberos is also an integral part of Microsoft Active Directory and is supported by Red Hat Directory Server and Red Hat IDM.

1.5. HOW DOES KERBEROS PROVIDE SSO FOR JBOSS EAP?

Kerberos provides desktop-based SSO by issuing tickets from a KDC for use by the client and server. JBoss EAP can integrate with this existing process by using those same tickets in its own authentication and authorization process. Before trying to understand how JBoss EAP can reuse those tickets, it is best to first understand in greater detail how these tickets are issued as well as how authentication and authorization works with Kerberos in desktop-based SSO without JBoss EAP.

1.5.1. Authentication and Authorization with Kerberos in Desktop-Based SSO

To provide authentication and authorization, Kerberos relies on a third party, the KDC, to provide authentication and authorization decisions for clients accessing servers. These decisions happen in three steps:

1. Authentication exchange.
   When a principal first accesses the network or attempts to access a secured service without a ticket granting ticket (TGT), they are challenged to authenticate against the authentication service (AS) with their credentials. The AS validates the user's provided credentials against the configured identity store, and upon successful authentication, the principal is issued a TGT which is cached by the client. The TGT also contains some session information so future communication between the client and KDC is secured.

2. Ticket granting, or authorization, exchange.
   Once the principal has been issued a TGT, they may attempt to access secured services or resources. The principal sends a request to the ticket granting service (TGS), passing the TGT it was issued by the KDC and requesting a service ticket (ST) for a specific destination. The TGS checks the TGT provided by the principal and verifies they have proper permissions to access the requested resource. If successful, the TGS issues an ST for the principal to access that specific destination. The TGS also creates session information for both the client as well as the destination server to allow for secure communication between the two. This session information is encrypted separately such that the client and server can only decrypt its own session information using long-term keys separately provided by the KDC to each, from previous transactions. The TGS then responds to the client with the ST which includes the session information for both the client and server.

3. Accessing the server.
   Now that the principal has an ST for the secured service as well as a mechanism for secure communication to that server, the client may now establish a connection and attempt to access the secured resource. The client starts by passing the ST to the destination server. This ST contains the server component of the session information which it received from the TGS for that

destination. The server attempts to decrypt the session information passed to it by the client, using its long-term key from the KDC. If it succeeds, the client has been successfully authenticated to the server and the server is also considered authenticated to the client. At this point, trust has been established and secured communication between the client and server may proceed.

NOTE

Despite the fact that unauthorized principals cannot actually use a TGT, a principal will only be issued a TGT after they first successfully authenticate with the AS. Not only does this ensure that only properly authorized principals are ever issued a TGT, it also reduces the ability for unauthorized third parties to obtain TGTs in an attempt to compromise or exploit them, for example using offline dictionary or brute-force attacks.

1.5.2. Kerberos and JBoss EAP

JBoss EAP can integrate with an existing Kerberos desktop-based SSO environment to allow for those same tickets to provide access to web applications hosted on JBoss EAP instances. In a typical setup, a JBoss EAP instance would be configured to use Kerberos authentication with SPNEGO using either the legacy **`security`** subsystem or the **`elytron`** subsystem. An application, configured to use SPNEGO authentication, is deployed to that JBoss EAP instance. A user logs in to a desktop, which is governed by Kerberos, and completes an authentication exchange with the KDC. The user then attempts to access a secured resource in the deployed application on that JBoss EAP instance directly using a web browser. JBoss EAP responds that authorization is required to access the secured resource. The web browser obtains the user's TGT ticket and then performs the ticket granting, or authorization, exchange with the KDC to validate the user and obtain a service ticket. Once the ST is returned to the browser, it wraps the ST in a request formatted for SPNEGO, and sends it back to the web application running on JBoss EAP. JBoss EAP then unpacks the SPNEGO request and performs the authentication using the either the legacy **`security`** subsystem or **`elytron`** subsystem. If the authentication succeeds, the user is granted access to the secured resource.

CHAPTER 2. HOW TO SET UP SSO FOR JBOSS EAP WITH KERBEROS

2.1. REQUIRED COMPONENTS

You must have the following components when setting up JBoss EAP for SSO with Kerberos:

- A properly configured Kerberos environment
- A JBoss EAP instance
- A web application

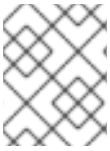
2.1.1. About JBoss Negotiation Toolkit

The JBoss Negotiation Toolkit is a debugging tool to help users debug and test authentication mechanisms before introducing an application into production. It is an unsupported tool but can be very helpful, as SPNEGO can be difficult to configure for web applications.

You can download a prebuilt WAR file of the JBoss Negotiation Toolkit from the JBoss Negotiation Toolkit repository. You should download the version of JBoss Negotiation Toolkit that matches the version of JBoss Negotiation included in JBoss EAP. For example, if you are using JBoss EAP 7.1, which uses JBoss Negotiation **`3.0.4.Final-redhat-1`**, you should use **`jboss-negotiation-toolkit-3.0.4.Final.war`**. You can determine which version of JBoss Negotiation is being used by looking at ***`EAP_HOME`*`/modules/system/layers/base/org/jboss/security/negotiation/main/module.xml`**.

2.2. KERBEROS ENVIRONMENT

As discussed in How Does Kerberos Provide SSO for JBoss EAP?, Kerberos relies on a third party, the KDC, to provide authentication and authorization decisions. This also requires clients, for example browsers, and their host to be properly configured to authenticate with the KDC. This guide is primarily focused on how to configure JBoss EAP and its hosted web applications, so configuring the KDC and Kerberos domain are not in the scope of this document.



NOTE

The subsequent sections assume a KDC and Kerberos domain have already been set up and properly configured.

2.3. DIFFERENCES FROM CONFIGURING PREVIOUS VERSIONS JBOSS EAP

There are a few noticeable differences between JBoss EAP 7.2 and earlier versions:

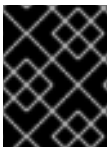
- The NegotiationAuthenticator valve is no longer required in the **`jboss-web.xml`**, but there still must be **`<security-constraint>`** and **`<login-config>`** elements defined in the **`web.xml`**. These are used to decide which resources are secured.

- The **auth-method** element in the **<login-config>** element is now a comma-separated list. The *exact* value **SPNEGO** must be there and should appear first in that list. In cases where **FORM** authentication is desired as a fallback, the *exact* value would be **SPNEGO,FORM**.
- The **jboss-deployment-structure.xml** file is not required when using the **elytron** subsystem.

2.4. CONFIGURING THE JBOSS EAP INSTANCE

You can configure an application deployed to JBoss EAP to use either **elytron** or the legacy **security** subsystem, but you cannot configure it to use both.

2.4.1. Configure the Elytron Subsystem



IMPORTANT

The following steps assume you have a working KDC, Kerberos domain, and browsers configured and working.

1. Configure a **kerberos-security-factory**.

```
/subsystem=elytron/kerberos-security-
factory=krbSF:add(principal="HTTP/host@REALM",
path="/path/to/http.keytab", mechanism-oids=[1.2.840.113554.1.2.2,
1.3.6.1.5.5.2])
```

2. Configure the system properties for Kerberos.
Depending on how your environment is configured, you will need to set some of the system properties below.

System Property	Description
java.security.krb5.kdc	The host name of the KDC.
java.security.krb5.realm	The name of the realm.
java.security.krb5.conf	The path to the configuration **krb5.conf** file.
sun.security.krb5.debug	If **true**, debugging mode will be enabled.

To configure a system property in JBoss EAP using the management CLI:

```
/system-
property=java.security.krb5.conf:add(value="/path/to/krb5.conf")
```

3. Configure an Elytron security realm for assigning roles.
The client's Kerberos token will provide the principal, but you need a way to map that principal to a role for your application. There are several ways to accomplish this, but this example creates a **filesystem-realm**, adds a user to the realm that matches the principal from the Kerberos token, and assigns roles to that user.

IMPORTANT

`filesystem-realm` is provided as Technology Preview only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

```
/subsystem=elytron/filesystem-realm=exampleFsRealm:add(path=fs-
realm-users, relative-to=jboss.server.config.dir)

/subsystem=elytron/filesystem-realm=exampleFsRealm:add-
identity(identity=user1@REALM)

/subsystem=elytron/filesystem-realm=exampleFsRealm:add-identity-
attribute(identity=user1@REALM, name=Roles, value=["Admin","Guest"])
```

4. Add a **`simple-role-decoder`**.

```
/subsystem=elytron/simple-role-decoder=from-roles-
attribute:add(attribute=Roles)
```

This **`simple-role-decoder`** decodes a principal's roles from the **`Roles`** attribute. You can change this value if your roles are in a different attribute.

5. Configure a **`security-domain`**.

```
/subsystem=elytron/security-domain=exampleFsSD:add(realms=
[{realm=exampleFsRealm, role-decoder=from-roles-attribute}],
default-realm=exampleFsRealm, permission-mapper=default-permission-
mapper)
```

6. Configure an **`http-authentication-factory`** that uses the **`kerberos-security-factory`**.

```
/subsystem=elytron/http-authentication-factory=example-krb-http-
auth:add(http-server-mechanism-factory=global, security-
domain=exampleFsSD, mechanism-configurations=[{mechanism-
name=SPNEGO, mechanism-realm-configurations=[{realm-
name=exampleFsSD}], credential-security-factory=krbSF}])
```

7. Configure an **`application-security-domain`** in the **`undertow`** subsystem:

```
/subsystem=undertow/application-security-domain=app-spnego:add(http-
authentication-factory=example-krb-http-auth)
```

2.4.2. Configure the Legacy Security Subsystem

JBoss EAP comes with all the components necessary to use Kerberos, using SPNEGO and JBoss Negotiation, for SSO with deployed applications, but the following configuration changes need to be made:

NOTE

The management CLI commands shown assume that you are running a JBoss EAP standalone server. For more details on using the management CLI for a JBoss EAP managed domain, see the JBoss EAP *Management CLI Guide*.

1. Configure the server identity, or host, security domain.
   This security domain authenticates the container itself to the KDC. It needs to use a login module which accepts a static login mechanism since a real user is not involved in this connection. The following example uses a static principal and references a keytab file which contains the credential.

   Example: Creating a Server Identity Security Domain

   ```
   /subsystem=security/security-domain=host:add(cache-type=default)

   /subsystem=security/security-
   domain=host/authentication=classic:add()

   /subsystem=security/security-
   domain=host/authentication=classic/login-
   module=Kerberos:add(code=Kerberos, flag=required, module-options=
   [storeKey=true, refreshKrb5Config=true, useKeyTab=true,
   principal=host/testserver@MY_REALM,
   keyTab=/home/username/service.keytab, doNotPrompt=true,
   debug=false])

   reload
   ```

   If using the IBM JDK, the options for Kerberos module are different. The **`jboss.security.disable.secdomain.option`** system property must be set to **`true`**. See Configure relevant system properties for more information. In addition, the login module should be configured as follows:

   Example: IBM JDK

   ```
   /subsystem=security/security-domain=host:add(cache-type=default)

   /subsystem=security/security-
   domain=host/authentication=classic:add()

   /subsystem=security/security-
   domain=host/authentication=classic/login-
   module=Kerberos:add(code=Kerberos, flag=required, module-options=
   [principal=host/testserver@MY_REALM, keyTab="file:///root/keytab",
   credsType=acceptor])

   reload
   ```

For a complete list of options for configuring the **Kerberos** login module, see the JBoss EAP *Login Module Reference*.

2. Configure the web application security domain.
The web application security domain is used to authenticate the individual user to the KDC. There needs to be at least one login module to authenticate the user. There also must be a way to search for the roles to apply to the user. This can be accomplished in many different ways, for example by adding a **<mapping>** that manually maps users to roles, adding a second login module for mapping users to roles, and so on.

   The following shows an example web application security domain.

   Example: Creating a Server Identity Security Domain

```
/subsystem=security/security-domain=app-spnego:add(cache-
type=default)

/subsystem=security/security-domain=app-
spnego/authentication=classic:add()

/subsystem=security/security-domain=app-
spnego/authentication=classic/login-module=SPNEGO:add(code=SPNEGO,
flag=required, module-options=[serverSecurityDomain=host])

reload
```

   For a complete list of options for configuring the **SPNEGO** login module, see the JBoss EAP *Login Module Reference*.

3. Configure relevant system properties.
JBoss EAP offers the ability to configure system properties related to connecting to Kerberos servers. Depending on the KDC, Kerberos Domain, and network configuration, the following system properties may or may not be required.

```
<system-properties>
  <property name="java.security.krb5.kdc" value="mykdc.mydomain"/>
  <property name="java.security.krb5.realm" value="MY_REALM"/>
  <property name="java.security.krb5.conf"
value="/path/to/krb5.conf"/>
  <property name="jboss.security.disable.secdomain.option"
value="true"/>
  <property name="sun.security.krb5.debug" value="false"/>
</system-properties>
```

Property	Description
java.security.krb5.kdc	The host name of the KDC.
java.security.krb5.realm	The name of the realm.
java.security.krb5.conf	The path to the configuration **krb5.conf** file.

Property	Description
jboss.security.disable.secdomain.option	When set to **true**, disables automatic adding of **jboss.security.security_domain** login module option to login modules declared in the security domain. Must be set to **true** when using the IBM JDK.
sun.security.krb5.debug	If **true**, debugging mode will be enabled.

NOTE

By default, each login module defined in a security domain has the **jboss.security.security_domain** module option added to it automatically. This option causes problems with login modules which check to make sure that only known options are defined. The IBM Kerberos login module, **com.ibm.security.auth.module.Krb5LoginModule** is one of these. This behavior of adding this module option can disabled by setting the **jboss.security.disable.secdomain.option** system property to **true** when starting JBoss EAP. This can be accomplished by configuring the **<system-properties>**, using the management CLI or management console, or by adding **-Djboss.security.disable.secdomain.option=true** to the startup parameters.

For more information about configuring system properties, see the JBoss EAP *Management CLI Guide*.

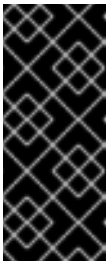
2.5. CONFIGURING THE WEB APPLICATION

Once the security domains have been configured, the web application must be configured to use those security domains in order to enable Kerberos authentication. Once the application changes have been made, it can be deployed to the JBoss EAP instance and begin using Kerberos for authentication.

The following updates must be made the application:

1. Configure the **web.xml** to use the SPNEGO authentication method.
   The **web.xml** file should contain the following:

   - A **<security-constraint>** with a **<web-resource-collection>** containing a **<url-pattern>** that maps to the URL pattern of the secured area. Optionally, **<security-constraint>** may also contain an **<auth-constraint>** stipulating the allowed roles.
   - If any roles were specified in the **<auth-constraint>**, those roles should be defined in a **<security-role>**.
   - A **<login-config>** containing a **<auth-method>** with the *exact* value of **SPNEGO**.



IMPORTANT

The **<auth-method>** element expects a comma-separated list of specific values. For **SPNEGO** authentication to be properly configured, the *exact* value **SPNEGO** must appear in the **<auth-method>** element and should appear first. Incorporating additional authentication types is discussed in Adding a FORM Login as a Fallback.

The **<security-constraint>** and **<security-role>** elements enable administrators to set up restricted or unrestricted areas based on URL patterns and roles. This allows resources to be secured or unsecured.

Example: `web.xml` File

```
<web-app>
  <display-name>App1</display-name>
  <description>App1</description>
  <!-- Define a security constraint that requires the Admin role
to access resources -->
  <security-constraint>
    <display-name>Security Constraint on Conversation</display-
name>
    <web-resource-collection>
      <web-resource-name>exampleWebApp</web-resource-name>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>Admin</role-name>
    </auth-constraint>
  </security-constraint>
  <!-- Define the Login Configuration for this Application -->
  <login-config>
    <auth-method>SPNEGO</auth-method>
    <realm-name>SPNEGO</realm-name>
  </login-config>
  <!-- Security roles referenced by this web application -->
  <security-role>
    <description>Role required to log in to the
Application</description>
    <role-name>Admin</role-name>
  </security-role>
</web-app>
```

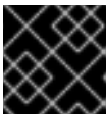
2. Configure **`jboss-web.xml`** to use the configured security domain.
   The **`jboss-web.xml`** file should have the following:

   - A **<security-domain>** to specify which security domain to use for authentication and authorization.
   - Optionally **<jacc-star-role-allow>**, which enables the use of the asterisk character in role-name element in **`web.xml`** to match multiple role names.

   Example: `jboss-web.xml` File

```
<jboss-web>
  <security-domain>app-spnego</security-domain>
  <jacc-star-role-allow>true</jacc-star-role-allow>
</jboss-web>
```

3. Add the JBoss Negotiation dependencies to the deployment for the legacy **`security`** subsystem.



IMPORTANT

If you are using the **`elytron`** subsystem, you can skip this step.

A web application using SPNEGO and JBoss Negotiation requires a dependency to be defined in **`jboss-deployment-structure.xml`** so that the JBoss Negotiation classes can be located. Since JBoss EAP provides all necessary JBoss Negotiation and related classes, the application just needs to declare them as dependencies to use them.

Using `jboss-deployment-structure.xml` to Declare Dependencies

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.jboss.security.negotiation"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

Alternatively, this dependency may be defined in a **`META-INF/MANIFEST.MF`** file instead:

Using `META-INF/MANIFEST.MF` to Declare Dependencies

```
Manifest-Version: 1.0
Dependencies: org.jboss.security.negotiation
```

2.6. ADDITIONAL CONSIDERATIONS FOR ACTIVE DIRECTORY

This section describes how to configure the accounts required for SPNEGO authentication to be used when JBoss EAP is running on a Microsoft Windows server, which is a part of the Active Directory domain.

In this section, the host name that is used to access the server as is referred to as ***`HOST_NAME`***, the realm is referred to as ***`REALM`***, the domain is referred to as ***`DOMAIN`***, and the server hosting the JBoss EAP instance is referred to as ***`MACHINE_NAME`***.

2.6.1. Configuration for Microsoft Windows Domain

1. Clear existing service principal mappings.
   On a Microsoft Windows network some mappings are created automatically. Delete the automatically created mappings to map the identity of the server to the service principal for negotiation to take place correctly. The mapping enables the web browser on the client computer to trust the server and attempt SPNEGO. The client computer verifies with the domain controller for a mapping in the form of HTTP/***`HOST_NAME`***.

The following are the steps to delete the existing mappings:

List the mapping registered with the domain for the computer using the command:

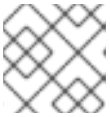
```
setspn -L MACHINE_NAME
```

Delete the existing mappings using the commands:

```
setspn -D HTTP/HOST_NAME MACHINE_NAME
```

```
setspn -D host/HOST_NAME MACHINE_NAME
```

2. Create a host user account.



NOTE

Ensure the host user name is different from the ***MACHINE_NAME***.

In the rest of the section the host user name is referred to as ***USER_NAME***.

3. Define the mapping between the ***USER_NAME*** and ***HOST_NAME***.
Run the following command to configure the service principal mapping:

```
ktpass -princ HTTP/HOST_NAME@REALM -pass * [-kvno 0] -mapuser
DOMAIN\USER_NAME -out jboss.keytab -ptype KRB5_NT_PRINCIPAL -crypto
all
```

Enter the password for the user name, when prompted.

Verify the mapping by running the following command: **`setspn -L USER_NAME`**.

NOTE

If you get **`KrbException: Specified version of key is not available`** errors from the JRE, you might need to set the Key Version to **`0`** : **`-kvno 0`** . Note that *REALM* needs to be all in uppercase, while the *HOST_NAME* should be all lowercase. Also, the *HOST_NAME* must be a FQDN, and must be a resolvable **`A`** or **`AAAA`** record, but not a **`CNAME`** record.

Using **`-crypto all`** only works on Windows Server 2008 and later. For Windows Server 2003, you must specify the exact setting.

4. Define the principal within the security domain.
The principal can be defined or updated in the **`elytron`** or legacy **`security`** subsystem to **`HTTP/HOST_NAME@REALM`**.

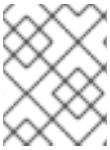
IMPORTANT

If you make any modification to any user, for example changing options or their password, you must regenerate the **`keytab`**.

CHAPTER 3. ADDITIONAL FEATURES

3.1. ADDING A FORM LOGIN AS A FALLBACK

JBoss EAP and applications deployed to it can also configure a FORM login authentication mechanism to use as a fallback. This allows applications to present a login page for authentication in cases where Kerberos/SPNEGO tokens are not present. This authentication happens independent of the Kerberos authentication. As a result, depending on how the FORM login fallback is configured, users may require separate credentials to authenticate by this method.



NOTE

The fallback to FORM login is available when no SPNEGO or NTLM tokens are present or, when a SPNEGO token is present, but from another KDC.

3.1.1. Update Your Application

The following steps are required to configure your application for FORM login as a fallback:

1. Configure JBoss EAP and the web application to use Kerberos and SPNEGO.
   See How to Set Up SSO for JBoss EAP with Kerberos for the steps required to configure JBoss EAP and web applications to use Kerberos and SPNEGO for authentication and authorization.

2. Add the login and error pages.
   To use FORM login, a login and error page are required. These files are added to web application and are used in the authentication process.

   Example: `login.jsp` File

```
<html>
  <head></head>
  <body>
    <form id="login_form" name="login_form" method="post"
action="j_security_check" enctype="application/x-www-form-
urlencoded">
      <center> <p>Please login to proceed.</p> </center>
      <div style="margin-left: 15px;">
        <p> <label for="username">Username</label> <br /> <input
id="username" type="text" name="j_username"/> </p>
        <p> <label for="password">Password</label> <br /> <input
id="password" type="password" name="j_password" value=""/> </p>
        <center> <input id="submit" type="submit" name="submit"
value="Login"/> </center>
      </div>
    </form>
  </body>
</html>
```

   Example: `error.jsp` File

```
<html>
  <head></head>
  <body>
```

```
    <p>Login failed, please go back and try again.</p>
  </body>
</html>
```

3. Modify the **web.xml**.
   After adding the login and error pages to the web application, the **web.xml** must be updated to use these files for FORM login. The *exact* value **FORM** must be added to the **<auth-method>** element. Since **<auth-method>** expects a comma-separated list and order is significant, the *exact* value for **<auth-method>** must be updated to **SPNEGO,FORM**. In addition, a **<form-login-config>** element must be added to **<login-config>** and the paths to the login and error pages specified as **<form-login-page>** and **<form-error-page>** elements.

   Example: Updated web.xml File

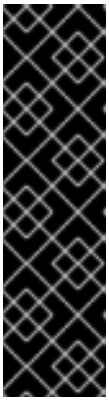
```
<web-app>
  <display-name>App1</display-name>
  <description>App1</description>
  <!-- Define a security constraint that requires the Admin role to
access resources -->
  <security-constraint>
    <display-name>Security Constraint on Conversation</display-name>
    <web-resource-collection>
      <web-resource-name>examplesWebApp</web-resource-name>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>Admin</role-name>
    </auth-constraint>
  </security-constraint>
  <!-- Define the Login Configuration for this Application -->
  <login-config>
    <auth-method>SPNEGO,FORM</auth-method>
    <realm-name>SPNEGO</realm-name>
    <form-login-config>
      <form-login-page>/login.jsp</form-login-page>
      <form-error-page>/error.jsp</form-error-page>
    </form-login-config>
  </login-config>
  <!-- Security roles referenced by this web application -->
  <security-role>
    <description> role required to log in to the
Application</description>
    <role-name>Admin</role-name>
  </security-role>
</web-app>
```

3.1.2. Update the Elytron Subsystem

1. Add a mechanism for **FORM** authentication in the **http-authentication-factory**.
   You can use the existing **http-authentication-factory** you configured for kerberos-based authentication and an additional mechanism for **FORM** authentication.

```
/subsystem=elytron/http-authentication-factory=example-krb-http-
auth:list-add(name=mechanism-configurations, value={mechanism-
name=FORM})
```

2. Add additional fallback principals.
   The existing configuration for kerberos-based authentication should already have a security realm configured for mapping principals from kerberos token to roles for the application. You can add additional users for fallback authentication to that realm. For example if you used a **`filesystem-realm`**, you can simply create a new user with the appropriate roles:



IMPORTANT

`filesystem-realm` is provided as Technology Preview only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

```
/subsystem=elytron/filesystem-realm=exampleFsRealm:add-
identity(identity=fallbackUser1)

/subsystem=elytron/filesystem-realm=exampleFsRealm:set-
password(identity=fallbackUser1, clear={password="password123"})

/subsystem=elytron/filesystem-realm=exampleFsRealm:add-identity-
attribute(identity=fallbackUser1, name=Roles, value=
["Admin","Guest"])
```

3.1.3. Update the Legacy Security Subsystem

If you are using the legacy **`security`** subsystem in JBoss EAP, you must update the security domain for fallback authentication.

The web application security domain must be configured to support a fallback login mechanism. This requires the following steps:

1. Add a new security domain to serve as a fallback authentication method.
2. Add a **`usernamePasswordDomain`** module option to the web application security domain that points to the fallback domain.

Example: Security Domain Configured with a Fallback Security Domain

```
/subsystem=security/security-domain=app-fallback:add(cache-type=default)

/subsystem=security/security-domain=app-
fallback/authentication=classic:add()

/subsystem=security/security-domain=app-
```

```
fallback/authentication=classic/login-
module=UsersRoles:add(code=UsersRoles, flag=required, module-options=
[usersProperties="file:${jboss.server.config.dir}/fallback-
users.properties",
rolesProperties="file:${jboss.server.config.dir}/fallback-
roles.properties"])

/subsystem=security/security-domain=app-
spnego/authentication=classic/login-module=SPNEGO:add(code=SPNEGO,
flag=required, module-options=[serverSecurityDomain=host])

/subsystem=security/security-domain=app-
spnego/authentication=classic/login-module=SPNEGO:map-put(name=module-
options, key=usernamePasswordDomain, value=app-fallback)

/subsystem=security/security-domain=app-
spnego/authentication=classic/login-module=SPNEGO:map-put(name=module-
options, key=password-stacking, value=useFirstPass)

reload
```

3.2. SECURING THE MANAGEMENT INTERFACES WITH KERBEROS

In addition to providing Kerberos authentication in security domains, JBoss EAP also provides the ability to secure the management interfaces using Kerberos.

3.2.1. Secure the Management Interfaces with Kerberos Using Elytron

To configure Kerberos authentication for the HTTP management interface:

1. Follow the same instructions for configuring Kerberos authentication for applications to create an **`http-authentication-factory`** that does Kerberos authentication.

IMPORTANT

When configuring Kerberos authentication with the management interfaces, it is very important that you pay close attention to the service principal you configure for JBoss EAP to authenticate against the KDC. This service principal takes the form of **`service-name/hostname`**. JBoss EAP expects **`HTTP`** to be the service name, for example **`HTTP/localhost`**, when authenticating against the web-based management console and **`remote`** to be the service name, for example **`remote/localhost`**, for the management CLI.

2. Update the management HTTP interface to use the **`http-authentication-factory`**.

```
/core-service=management/management-interface=http-interface:write-
attribute(name=http-authentication-factory, value=example-krb-http-
auth)
```

To configure Kerberos authentication for SASL authentication for the management CLI:

1. Follow the same instructions for configuring Kerberos authentication for applications to create a security domain and **`kerberos-security-factory`**.

2. Add **GSSAPI** to the **configurable-sasl-server-factory**.

```
/subsystem=elytron/configurable-sasl-server-factory=configured:list-
add(name=filters, value={pattern-filter=GSSAPI})
```

3. Create a **sasl-authentication-factory** that uses the security domain and **kerberos-security-factory**.

Example: sasl-authentication-factory

```
/subsystem=elytron/sasl-authentication-factory=example-sasl-
auth:add(sasl-server-factory=configured, security-
domain=exampleFsSD, mechanism-configurations=[{mechanism-
name=GSSAPI, mechanism-realm-configurations=[{realm-
name=exampleFsSD}], credential-security-factory=krbSF}])
```

4. Update the management SASL interface to use the **sasl-authentication-factory**.

Example: Update sasl-authentication-factory

```
/core-service=management/management-interface=http-interface:write-
attribute(name=http-upgrade.sasl-authentication-factory,
value=example-sasl-auth)

reload
```

3.2.2. Secure the Management Interfaces With Kerberos Using Legacy Core Management Authentication

To enable Kerberos authentication on the management interfaces using legacy core management authentication, the following steps must be performed:

NOTE

The management CLI commands shown assume that you are running a JBoss EAP standalone server. For more details on using the management CLI for a JBoss EAP managed domain, see the JBoss EAP *Management CLI Guide*.

1. Enable the relevant system properties.
As discussed in a previous section, enable any needed JBoss EAP system properties for connecting to the Kerberos server.

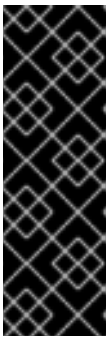
2. Add the Kerberos server identity to the security realm.
Before Kerberos authentication can be used in a security realm, a connection to a Kerberos server must be added. The following example shows how to add a Kerberos server identity to the existing Management Realm. You will need to replace **service-name**, **hostname**, and **MY-REALM** with the appropriate values.

Example CLI for Adding a Server Identity to a Security Realm

```
/core-service=management/security-realm=ManagementRealm/server-
identity=kerberos:add()
```

```
/core-service=management/security-realm=ManagementRealm/server-
identity=kerberos/keytab=service-name\/hostname@MY-
REALM:add(path=/home\/username\/service.keytab, debug=true)

reload
```



IMPORTANT

When configuring Kerberos authentication with the management interfaces, it is very important that you pay close attention to the service principal you configure for JBoss EAP to authenticate against the KDC. This service principal takes the form of **`service-name/hostname`**. JBoss EAP expects **`HTTP`** to be the service name, for example **`HTTP/localhost`**, when authenticating against the web-based management console and **`remote`** to be the service name, for example **`remote/localhost`**, for the management CLI.

3. Update the authentication method in the security realm.
   Once the Kerberos server identity has been properly configured, the authentication method in the security realm needs to be updated to use it.

   Example: Adding Kerberos Authentication to a Security Realm

```
/core-service=management/security-
realm=ManagementRealm/authentication=kerberos:add()

reload
```

IMPORTANT

Based on the order in which you have the authentication mechanisms defined in the security realm, JBoss EAP will attempt to authenticate the user in that order when accessing the management interfaces.

4. Secure both interfaces with Kerberos.
   In cases where you would like to secure both the web-based management console and management CLI with Kerberos, you need a Kerberos server identity configured for each. To add an additional identity, use the following command.

```
/core-service=management/security-realm=ManagementRealm/server-
identity=kerberos/keytab=remote\/hostname@MY-
REALM:add(path=/home\/username\/remote.keytab, debug=true)

reload
```

3.2.3. Connecting to the Management Interface

Before attempting to connect to the management interfaces, you need to have a valid Kerberos ticket. If the security realm fails to authenticate a user via Kerberos, when using the legacy security solution, it will attempt to authenticate the user using any of the subsequent methods specified in the **`<authentication>`** element. The **`elytron`** subsystem behaves similar to the legacy security solution.

If the Kerberos authentication mechanism fails, authentication falls back to any other mechanism that you have defined in the authentication factory that is protecting the management interface. Usually, DIGEST or BASIC is used as a fallback.

When you connect to the web-based management console using a browser, the security realm will attempt to authenticate you based on that ticket.

When connecting to the management CLI, you will need to use the `-Djavax.security.auth.useSubjectCredsOnly=false` parameter, as this allows the GSSAPI implementation to make use of the identity managed at the operating system level. You may also need to use the following parameters based on how your environment is set up:

-Djava.security.krb5.realm=*REALM_NAME*

Specifies the realm name.

-Djava.security.krb5.kdc=*KDC_HOSTNAME*

Specifies the location of the KDC.

--no-local-auth

Disables local authentication. This is useful if you are attempting to connect to a JBoss EAP instance running on the same machine you are running the script from.

Example Command

```
$ EAP_HOME/bin/jboss-cli.sh -c -
Djavax.security.auth.useSubjectCredsOnly=false --no-local-auth
```

WARNING

If an HTTP proxy is used between the client and server, it must take care to not share authenticated connections between different authenticated clients to the same server. If this is not honored, then the server can easily lose track of security context associations. A proxy that correctly honors client to server authentication integrity will supply the `Proxy-support: Session- Based-Authentication` HTTP header to the client in HTTP responses from the proxy. The client must *not* utilize the SPNEGO HTTP authentication mechanism through a proxy unless the proxy supplies this header with the `401 Unauthorized` response from the server.

3.3. KERBEROS AUTHENTICATION INTEGRATION FOR REMOTING

In addition to using Kerberos for securing the management interfaces and web applications, you can also configure Kerberos authentication for services accessed via remoting, such as EJBs.

System properties for Kerberos also need to be configured. For more information, see Configure the Elytron Subsystem.

3.3.1. Kerberos Authentication Integration Using Legacy Security Realms

To configure Kerberos authentication, you will need to do the following:

1. Configure a security domain with remoting and **`RealmDirect`**
   You need to configure a security domain for use by the service that is accessed by remoting. This security domain needs to make use of both the **`Remoting`** login module as well as a **`RealmDirect`** login module, such as **`RealmDirect`** or **`RealmUsersRoles`**. Essentially, it should look very similar to the **`other`** security domain provided by default. For more details on the specific configuration options of each login module, see the JBoss EAP *Login Module Reference*.

   Example: Security Domain with Remoting and RealmDirect Login Modules

   ```
   /subsystem=security/security-domain=krb-remoting-domain:add()

   /subsystem=security/security-domain=krb-remoting-
   domain/authentication=classic:add()

   /subsystem=security/security-domain=krb-remoting-
   domain/authentication=classic/login-
   module=Remoting:add(code=Remoting, flag=optional, module-options=
   [password-stacking=useFirstPass])

   /subsystem=security/security-domain=krb-remoting-
   domain/authentication=classic/login-
   module=RealmDirect:add(code=RealmDirect, flag=required, module-
   options=[password-stacking=useFirstPass, realm=krbRealm])

   /subsystem=security/security-domain=krb-remoting-
   domain/mapping=classic:add()

   /subsystem=security/security-domain=krb-remoting-
   domain/mapping=classic/mapping-
   module=SimpleRoles:add(code=SimpleRoles, type=role, module-options=
   ["testUser"="testRole"])

   reload
   ```

2. Configure a security realm for Kerberos authentication.
   Setting up a security realm with Kerberos authentication is covered in the Securing the Management Interfaces with Kerberos section.

   Example: Security Realm

   ```
   /core-service=management/security-realm=krbRealm:add()

   /core-service=management/security-realm=krbRealm/server-
   identity=kerberos:add()

   /core-service=management/security-realm=krbRealm/server-
   identity=kerberos/keytab=remote\/localhost@JBOSS.ORG:add(path=\/path
   \/to\/remote.keytab, debug=true)

   /core-service=management/security-
   realm=krbRealm/authentication=kerberos:add(remove-realm=true)

   reload
   ```

3. Configure the HTTP connector in the **remoting** subsystem.
   In addition, you will need to configure the HTTP connector in the **remoting** subsystem to use the newly created security realm.

   Example: Remoting Subsystem

   ```
   /subsystem=remoting/http-connector=http-remoting-connector:write-
   attribute(name=security-realm, value=krbRealm)
   ```

4. Configure security for the service.
   You must also set up the service that is accessed using the remoting interface to **secured**. This will vary depending on the service. For example, with an EJB, you can use the **@SecurityDomain** and **@RolesAllowed** annotations.

3.3.2. Kerberos Authentication Integration Using Elytron

It is possible to define an Elytron security domain for Kerberos or GSSAPI SASL authentication for remoting authentication.

1. Define the security realm to load the identity from. It is used for assigning roles.

   ```
   /path=kerberos:add(relative-to=user.home, path=src/kerberos)

   /subsystem=elytron/properties-realm=kerberos-properties:add(users-
   properties={path=kerberos-users.properties, relative-to=kerberos,
   digest-realm-name=ELYTRON.ORG}, groups-properties={path=kerberos-
   groups.properties, relative-to=kerberos})
   ```

2. Define the Kerberos security factory for the server's identity.

   ```
   /subsystem=elytron/kerberos-security-factory=test-
   server:add(relative-to=kerberos, path=remote-test-server.keytab,
   principal=remote/test-server.elytron.org@ELYTRON.ORG)
   ```

3. Define the security domain and along with it, a SASL authentication factory.

   ```
   /subsystem=elytron/security-domain=KerberosDomain:add(default-
   realm=kerberos-properties, realms=[{realm=kerberos-properties, role-
   decoder=groups-to-roles}], permission-mapper=default-permission-
   mapper)

   /subsystem=elytron/sasl-authentication-factory=gssapi-
   authentication-factory:add(security-domain=KerberosDomain, sasl-
   server-factory=elytron, mechanism-configurations=[{mechanism-
   name=GSSAPI, credential-security-factory=test-server}])
   ```

4. Use the created **sasl-authentication-factory**, in the **remoting** subsystem, to enable it for remoting.

   Example CLI Command

```
/subsystem=remoting/http-connector=http-remoting-connector:write-
attribute(name=sasl-authentication-factory, value=gssapi-
authentication-factory)
```

5. Configure security for the service.
   If you reference the security domain in an EJB, you must specify the **`application-security-domain`** that maps to the Elytron security domain. For example, with an EJB, you can use the **`@SecurityDomain`** annotation.

   Example CLI Command

```
/subsystem=ejb3/application-security-
domain=KerberosDomain:add(security-domain=KerberosDomain)
```

With the new JBoss EAP 7.1 EJB client, the use of a JAAS Subject for identity association is no longer supported. Clients wishing to programmatically manage the Kerberos identity for an EJB call should migrate and use the **`AuthenticationConfiguration`** APIs directly, as follows:

```
// create your authentication configuration
AuthenticationConfiguration configuration =
AuthenticationConfiguration.empty()

.useProvidersFromClassLoader(SecuredGSSCredentialClient.class.getClassLoad
er())
    .useGSSCredential(getGSSCredential());

// create your authentication context
AuthenticationContext context =
AuthenticationContext.empty().with(MatchRule.ALL, configuration);

// create a callable that looks up an EJB and invokes a method on it
Callable<Void> callable = () -> {
...
};

// use your authentication context to run your callable
context.runCallable(callable);
```

The call to **`useGSSCredential(getGSSCredential())`** happens when creating the **`AuthenticationConfiguration`**. Client code that already has access to a JAAS Subject can easily be converted to obtain the **`GSSCredential`** as follows:

```
private GSSCredential getGSSCredential() {
    return Subject.doAs(subject, new PrivilegedAction<GSSCredential>() {

        public GSSCredential run() {
            try {
                GSSManager gssManager = GSSManager.getInstance();
                return
gssManager.createCredential(GSSCredential.INITIATE_ONLY);
            } catch (Exception e) {
                e.printStackTrace();
            }
            return null;
```

```
            }
        });
    }
```

Revised on 2019-01-25 07:15:19 UTC